

UNIVERSITY OF BUEA

P.O. Box 63,

Buea, South West Region

CAMEROON

Tel : (237) 3332 21 34/3332 26 90

Fax: (237) 3332 22 72



REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

COURSE: CEF 440 - Internet Programming and Mobile Programming

**TASK 6: Database design and Implementation
of the biometric student's attendance mobile
application**

Presented by:

NAMES	MATRICULES
DJEUNOU DJEUNOU MARIEKE JETTIE	FE21A168
KENNE DATEWO SUZY MAIVA	FE21A214
MATHO SONKWA HESTIE MAYELLE	FE21A438
SIAHA TOUKO AUBIN	FE21A304
TSAPZE ZAMBOU ROSELINE CYNTHIA	FE21A328

Academic year: 2023-2024

COURSE INSTRUCTOR: Dr. NKEMENI VALERY

Table of Contents

INTRODUCTION1

1. PURPOSE OF THE DATABASE2

1.1. Objective and use cases of the database2

1.2. Key requirements and constraints2

2. Conceptual, logical and physical models4

3. ER DIAGRAM5

3.1. Entities and relationships in the school5

3.2. Diagram in school:5

3.3. Entities and relationships in *AttendEase* mobile application6

3.4. Diagram in *AttendEase* system7

4. DATABASE SCHEMA8

5. IMPLEMENTATION10

CONCLUSION16

REFERENCES17

INTRODUCTION

In educational institutions, maintaining accurate and reliable attendance records is crucial for administrative purposes and ensuring student accountability. Traditional methods of attendance tracking, like manual roll calls or card-based systems, are prone to errors, fraud, and inefficiencies. A biometric student attendance system using fingerprint recognition offers a more secure, reliable, and automated solution. This report aims to design and implement a database system to support such an attendance tracking system using MongoDB, a non-relational database. The database is a critical component of the *AttendEase* biometric attendance system as it stores and manages all the essential data, including student information, fingerprint templates, attendance records, and administrative data. The integrity, scalability, and performance of the database directly impact the effectiveness of the attendance system. For this project, a single MongoDB database will be used the institutional database as well that will be integrated to ours in other to easily communicate with student's data in the institution. This database will leverage MongoDB's document-oriented structure to efficiently store and manage diverse data types, including biometric data. Using a single database simplifies the architecture, management, and ensures all related data is centralized for easy access and analysis.

1. PURPOSE OF THE DATABASE

1.1. Objective and use cases of the database

The primary objective of the database is to support the biometric student attendance system by: storing student profiles and fingerprint data, recording daily attendance logs, providing administrative tools for attendance management and reporting.

Use Cases:

- **Student Registration:** Store new student profiles and their fingerprint data.
- **Attendance Logging:** Record attendance entries each time a student scans their fingerprint.
- **Attendance Tracking:** Retrieve attendance records, date and time for monitoring and reporting.
- **Management Reports:** Generate reports on student attendance over specified periods.

1.2. Key requirements and constraints

Designing and implementing a biometric student attendance mobile application involves addressing various requirements and constraints to ensure the system is robust, efficient, and secure. This application will manage large volumes of student data and biometric records, necessitating careful consideration of several critical factors. Below is a detailed elaboration of these requirements and constraints.

1.2.1. Requirements

- **Scalability:** The application must handle a large number of students and frequent attendance records. It should scale horizontally and vertically to accommodate increasing data loads and user interactions.
- **Performance:** Quick retrieval and insertion of records are crucial. The system should be optimized for high-speed transactions to ensure real-time operations using techniques like indexing and caching.
- **Reliability:** Maintain data integrity and consistency. The system should implement mechanisms for data validation, error checking, and transactional integrity.
- **Security:** Protect sensitive biometric data with encryption and access controls. This includes using advanced encryption standards and strict access control policies.

1.2.2. Constraints:

- Data Size: Biometric data can be large, requiring efficient storage solutions.
- Consistency: Ensure no duplicate or fraudulent attendance records.

2. Conceptual, logical and physical models

Development of conceptual, logical, and physical models in the design of the database ensures an efficient, scalable, and secure system. MongoDB, as a NoSQL database, offers flexibility and scalability, making it suitable for managing diverse data types and handling large volumes of data effectively. Below, we explore each model in detail.

Conceptual Model:

At the conceptual level, the data model for the biometric student attendance system consists of the key entities and their relationships. The relationships between these entities define how the data is structured and accessed. For instance, each student can have multiple attendance records, and each attendance record is associated with a single student. The primary entities include:

- **Students:** Storing basic student information such as matricule, contact details, and enrolled courses.
- **Attendance Records:** Capturing the details of each student's attendance, including the date, time, and status (present, absent, late).
- **Fingerprint Templates:** Storing the unique fingerprint hash data for each student, which is used for identification and authentication during the attendance marking process.

Logical Model:

The logical data model translates the conceptual design into a more detailed representation, considering the specific data types, constraints, and relationships required by the chosen database technology. In this case, the system is being implemented using a non-relational database, specifically MongoDB, which follows a document-oriented data model.

- **Collections:** Students, Fingerprints, AttendanceRecords.
- **Documents:** JSON-like structures representing the entities and relationships.

Physical Model:

The physical data model represents the actual implementation of the database schema in the MongoDB environment. This includes the structure of the MongoDB documents, the indexing strategies, and any additional configurations or settings required to optimize the database performance and ensure data integrity.

- **MongoDB document collections with indexes on key fields** (e.g., student ID, timestamp for attendance records).

3. ER DIAGRAM

The entity relationship (ER) diagram provides a visual representation of the conceptual data model for the biometric student attendance system. It helps to clearly identify the key entities, their attributes, and the relationships between them.

3.1. Entities and relationships in the school

The main entities and relationships in the ER diagram of the institution are:

- **Admin:** He is responsible of managing the courses, students, and lecturers in the institution by performing CRUD (create, read, update, and delete) operations. He can then manage one or many courses, one or many lecturers, and one or many students. His attributes are: a unique identifier, an email, a name, and a password to secure his account.
- **Student:** This entity stores the basic information about each student enrolled in the school. Having attributes such as name, matricule, and email address, the student can be managed by one admin, he can be taught by one or many lecturers, and he is belonging to only one department under a faculty.
- **Lecturer:** He is managed by an admin, he belongs to one faculty and he teaches one or many students. We can have information of lecturers through the attributes such as his name, email address, and unique identifier.
- **Course:** a course can be created by the admin, and it is assigned to only one faculty. The attributes of a course are the course code and the course title
- **Faculty:** The attributes of a faculty are the faculty name and the location. The faculty can contain one or many students as well as one or many courses. This entity has a child entity, which is nothing else than the department, identified by the department name.

3.2. Diagram in school:

The ER diagram of the institution can be shown in the figure bellow.

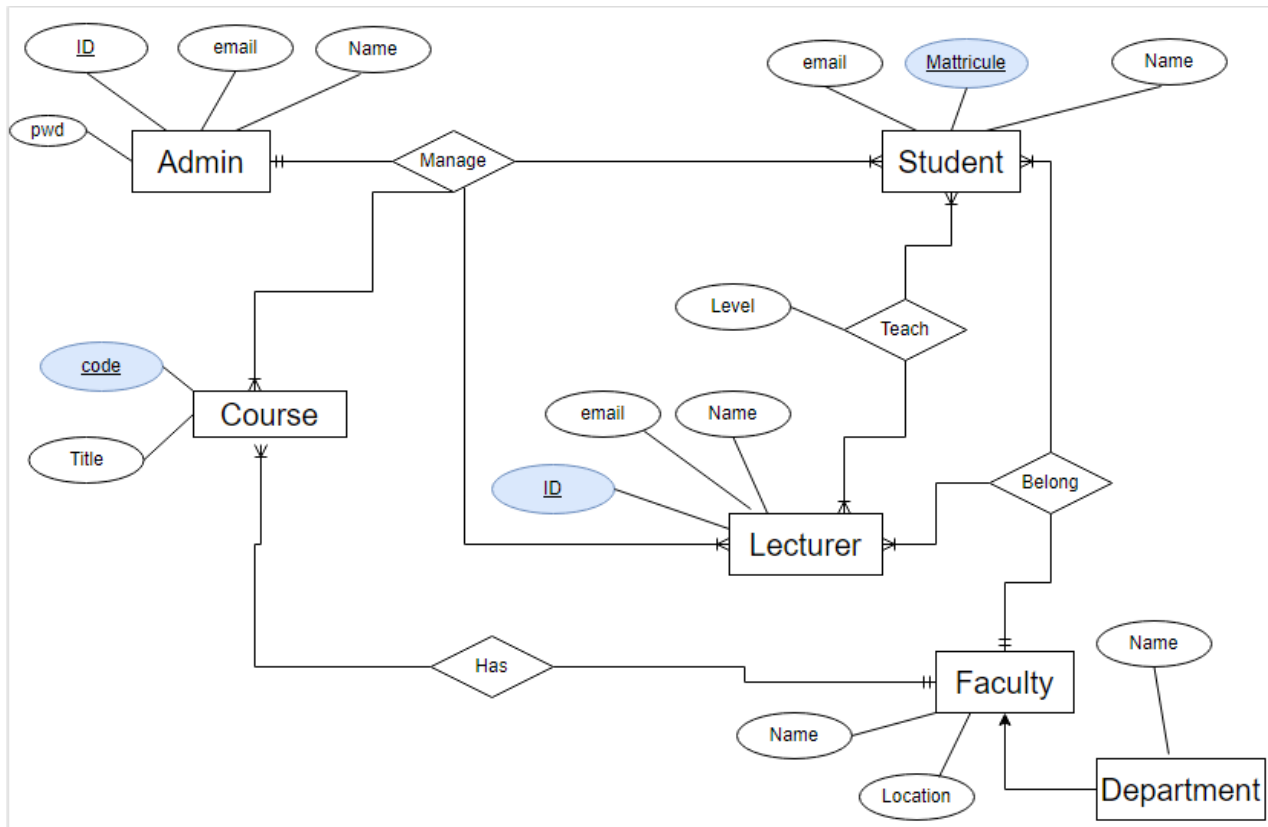


Figure: Anticipated ER diagram of the institution

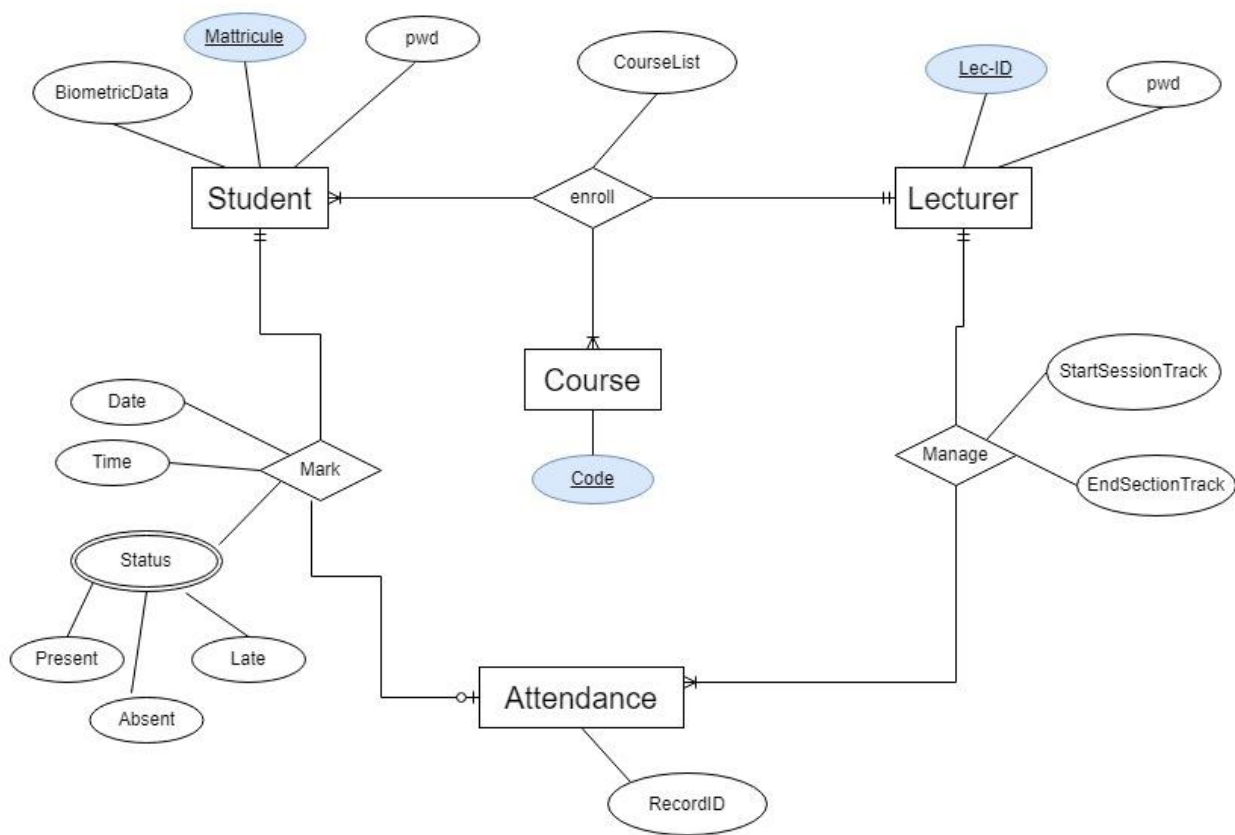
3.3. Entities and relationships in *AttendEase* mobile application

The main entities and relationships in the ER diagram are:

- **Student**: A student can enroll for one or many courses and a course is enrolled by many students. When he enrolled, a course list for student is created along. A student can either mark or not only one attendance, depending on if he is present to do so or not. So when he marks the attendance, the date and the time are tracked in other to determine the status of attendance. In our database, he can make use of his password to login toward the system, a biometric data or template is attached to his account to let him authenticate during the attendance marking process. He also has a matricule that uniquely identifies him, and we can keep track of his credentials (name, faculty, etc.) in the institutional database in the integration.
- **Lecturer**: The lecturer enrolls the courses he is teaching and he has the responsibility to manage the attendance. A lecturer enrolls one or many courses, and a course is enrolled by only one lecturer. He can also manage the attendance list by removing or adding a student attendance, export it, and the timeframe the attendance can be mark is counting.
- **Attendance or attendance Records**

- Attributes: RecordID
- This entity captures the attendance data for each student, including the date, time, and attendance status. An attendance can be marked by only one student, and a student does not necessary have to mark the attendance; an attendance record can be managed by one and only one lecturer, and a lecturer can manage many attendance.
- Course: courses are enrolled by many students and many lecturers

3.4. Diagram in *AttendEase* system



ER diagram of AttendEase mobile application

4. DATABASE SCHEMA

The database schema is critical for the design and implementation of a biometric student attendance mobile application. This schema involves organizing data into collections and defining relationships between them to ensure the system is efficient, scalable, and secure. Our diagrams below are drawn with the software draw.io

- **Collections and Documents in school**

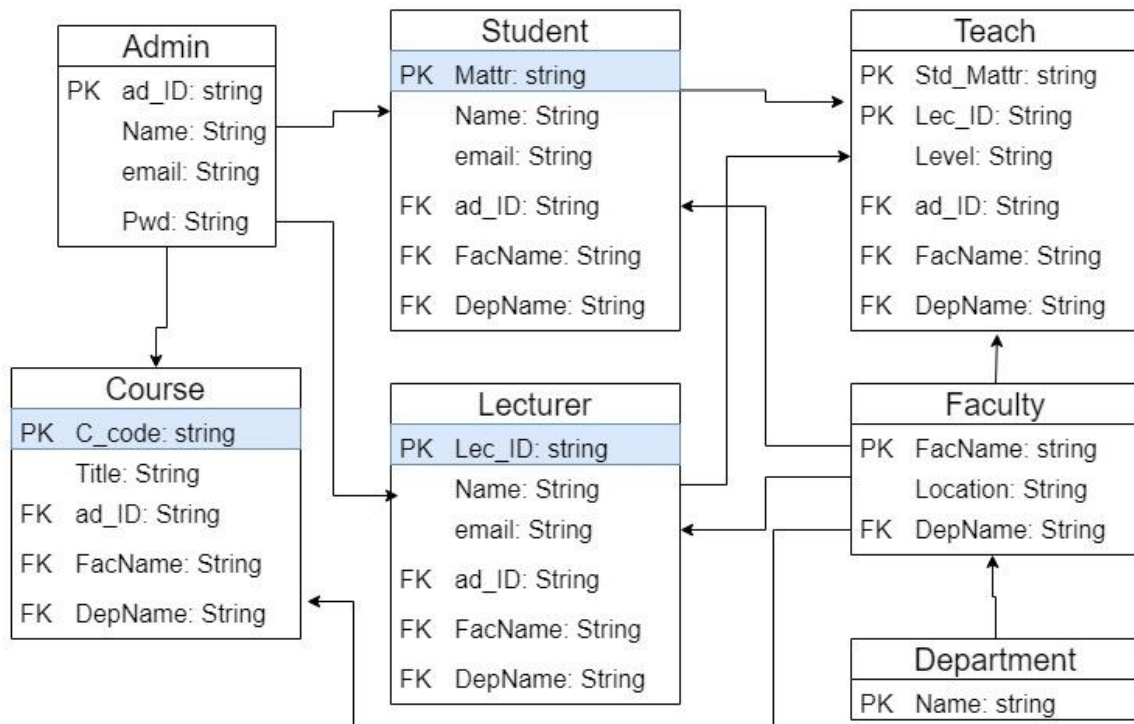


Figure: Anticipated Database schema of the institution

- **Collections and Documents in the *AttendEase* mobile application**

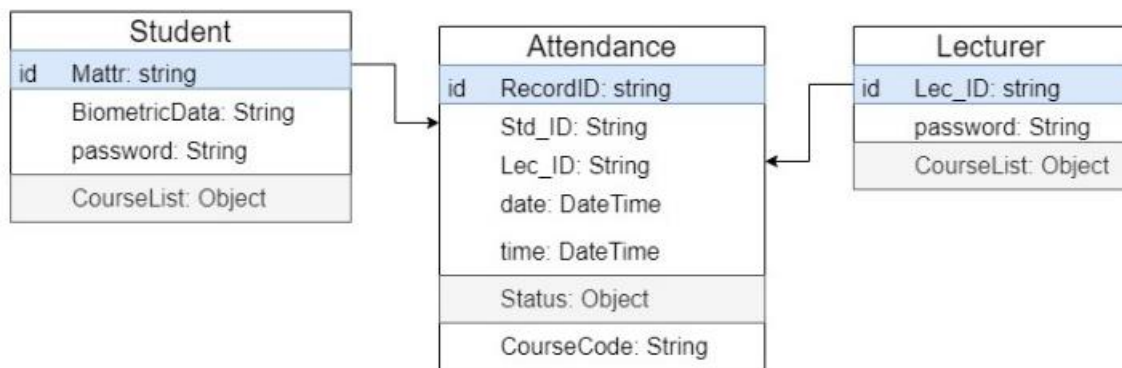


Figure: Database schema of AttendEase application

fingerprintTemplate or BiometricData example: "base64encodedtemplate"

CourseList object components: card or courses composed of course title and course code

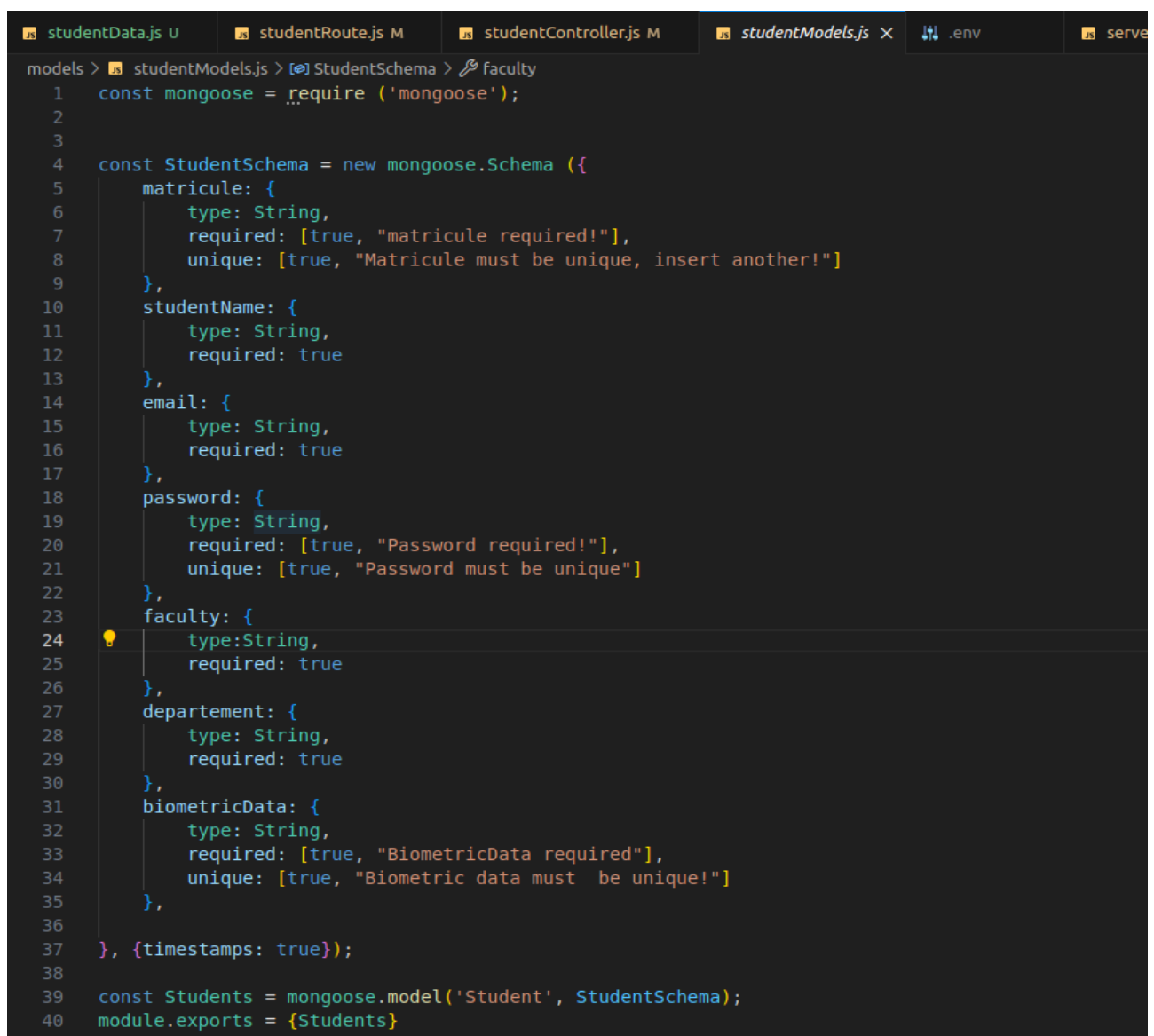
Status object: present, absent, late

5. IMPLEMENTATION

The implementation is done using mongodb database and we are dealing here with two databases: institutional that we have to integrate in our system and application database which is responsible to manage the attendance of students. This is achieved by making use of collections and documents in our system.

We started by installing mongoose via the npm install mongoose command then we created the mongoose models for each database. The database schema of our first database is designed to store information about students, instructors, faculty, departement , and others informationrelated to the school organizations.

Student schema



```
models > studentModels.js > StudentSchema > faculty
1  const mongoose = require ('mongoose');
2
3
4  const StudentSchema = new mongoose.Schema ({
5    matricule: {
6      type: String,
7      required: [true, "matricule required!"],
8      unique: [true, "Matricule must be unique, insert another!"]
9    },
10   studentName: {
11     type: String,
12     required: true
13   },
14   email: {
15     type: String,
16     required: true
17   },
18   password: {
19     type: String,
20     required: [true, "Password required!"],
21     unique: [true, "Password must be unique"]
22   },
23   faculty: {
24     type:String,
25     required: true
26   },
27   departement: {
28     type: String,
29     required: true
30   },
31   biometricData: {
32     type: String,
33     required: [true, "BiometricData required"],
34     unique: [true, "Biometric data must be unique!"]
35   },
36
37 }, {timestamps: true});
38
39 const Students = mongoose.model('Student', StudentSchema);
40 module.exports = {Students}
```

Instructor schema

```
studentData.js U studentRoute.js M studentController.js M instructorModels.js X
models > instructorModels.js > ...
1  const mongoose = require ('mongoose');
2
3
4  const InstructorSchema = new mongoose.Schema ({
5    instructorMatricule: {
6      type: String,
7      required: [true, "Matricule required!"],
8      unique: [true, "Matricule be unique"]
9    },
10   instructorName: {
11     type: String,
12     required: true
13   },
14   email: {
15     type: String,
16     required: [true, "Email address required"],
17     unique: true
18   },
19   password: {
20     type: String,
21     required: [true, "Password address required!"],
22     unique: [true, "Password must be unique!"]
23   }
24 }, {timestamps: true});
25
26
27 const Instructors = mongoose.model("Instructor", InstructorSchema);
28 module.exports = {Instructors};
29
```

Course schema

```
studentData.js U studentRoute.js M studentController.js M courseModels.js >
models > courseModels.js > [?] <unknown> > Courses
1  const mongoose = require ('mongoose');
2
3
4  const CourseSchema = new Schema ({
5    courseCode: {
6      type: String,
7      required: true
8    },
9    courseName: {
10     type: String,
11     required: true
12   },
13   courseLevel: {
14     type: String,
15     required: true
16   },
17   courseInstructor: []
18 }, {timestamps: true});
19
20 const Courses = mongoose.model("Course", CourseSchema);
21 module.exports = {Courses};
22
```

For the second database, we use the following schema:

```
models > attendanceModels.js > AttendanceSchema > studentName
1  const mongoose = require ('mongoose');
2
3
4  const AttendanceSchema = new mongoose.Schema ({
5      attendanceId: {
6          type: String,
7          required: true,
8      },
9      attendanceDate: {
10         type: Date,
11         required: true
12     },
13
14     attendancePeriod: {
15         type: String,
16         required: true,
17     },
18     studentName: {
19         type: String,
20         required: true
21     },
22 }, {timestamps: true});
23
24
25 const Attendances = mongoose.model("Attendance", AttendanceSchema);
26 module.exports = {Attendances}
```

After that, we create the different controllers and routes to insert data in the database. For example, here is the controller of student where we performed the CRUD operations via the POST, GET, DELETE AND UPDATE request.

```

controllers > studentController.js > getAllStudent > message
1  const express = require ('express');
2  const Student = require ("../models/studentModels");
3  const { default: mongoose } = require('mongoose');
4
5
6
7  //function to get all student based on their level
8  async function getAllStudent(req, res, next) {
9      try {
10         const students = await Student.find({});
11
12         return next(
13             res.status(200).json(students)
14         )
15     } catch (error) {
16         return next (
17             res.status(400).json({
18                 message: error
19             })
20         )
21     }
22 }
23
24
25 //function to get a student
26 async function getOneStudent(req, res, next) {
27     const {id}=req.params;
28     try {
29         const student = await Student.findById();
30
31         return next(
32             res.status(200).json(student)
33         )
34     } catch (error) {
35         return next (
36             res.status(400).json({
37                 message: error
38             })
39         )
40     }
41 }

```

```

controllers > studentController.js > getAllStudent > message
44 //function to create a student to the database
45 async function createStudent(req, res, next) {
46     const studentData = req.body;
47
48     try {
49         const newStudent = await Student.create(studentData);
50
51         return next (
52             res.status(200).json({
53                 status: "OK",
54                 message: "Student successfully added!"
55             })
56         )
57     } catch (error) {
58         res.status(400).json({
59             message: "Failed to create student!"
60         })
61     }
62 };
63
64
65 //function to update student
66 async function updateStudent(req, res, next) {
67     const {id} = req.params;
68
69     const update = req.body;
70
71     if( !mongoose.Types.ObjectId.isValid(id)){
72         return next(
73             res.status(404).json({
74                 message: "Invalid id!"
75             })
76         )
77     }
78
79     const student = await Student.findByIdAndUpdate({_id: id}, {
80         ...req.body, update
81     })
82     if(!student){
83         return next(
84             res.status(404).json({
85                 message: "student Not Found!"
86             })
87         )
88     }
89 }

```

To be able to send request and get a respond, we used routes to link the controller with the model that we have. See below the routes files of students.


```

routes > studentRoute.js > ...
1  const express = require("express");
2  const {getAllStudent, deleteStudent, updateStudent, createStudent, getOneStudent} = require(".
3
4  const router = express.Router();
5
6
7  /**
8   * Get all students
9   */
10 router.get("/", getAllStudent);
11
12
13 /**
14  * Get one student
15  */
16 router.get("/:id", getOneStudent);
17
18
19 /**
20  * Create a student
21  */
22
23 router.post("/", createStudent);
24
25 /**
26  * update student info
27  */
28 router.patch("/:id", updateStudent);
29
30 /**
31  * delete student
32  */
33
34 router.delete("/:id", deleteStudent);
35
36
37 module.export = {router};

```

Dependencies:

- Node.js
- Express
- MongoDB
- Mongoose
- Dotenv
- Postman

CONCLUSION

This report has outlined the detailed design and implementation of the database for the biometric student attendance system using MongoDB, a leading NoSQL database platform. The conceptual data model (ER diagram) identified the key entities along with their relationships. This provided a solid foundation for the logical and physical data model design, which leveraged MongoDB's document-oriented approach to efficiently store and manage the attendance data. One of the key advantages of using MongoDB for this application is its inherent scalability and flexibility. As the number of students and attendance records grows, the database can easily accommodate the increased data volumes and traffic without compromising performance. Additionally, the document-oriented data model allows for seamless integration with the fingerprint recognition module, enabling a streamlined and efficient biometric attendance tracking system.

REFERENCES

- [1] <https://ieeexplore.ieee.org/document/8560853>
- [2] <https://www.ajol.info/index.php/njt/article/view/216445/204145>
- [3]https://www.researchgate.net/publication/329481792_Design_and_Implementation_of_a_Student_Attendance_System_Using_Iris_Biometric_Recognition
- [4]https://www.academia.edu/36680026/Design_and_Implementation_of_student_attendance_authentication_system